Research on A Tunable Consistency Strategy of the Distributed Database

Chaozhi Yang
School of Information Science and Technology
Dalian Maritime University
Dalian, Liaoning, China 0411-84729544
Email: 18342208289@163.com

Tingting Cai
School of Information Science and Technology
Dalian Maritime University

Zhihuai Li
School of Information Science andTechnology
Dalian Maritime University
Dalian, Liaoning, China 0411-84727856
Email: qhlee@dlmu.edu.cn

***Abstract*—This paper proposes a Tunable Consistency Strategy(TCS) of the Distributed Databases. The TCS can divide the consistency dynamically for different requests. A request scheduling layer was added in the TCS to schedule different requests according to user's demand. The TCS can coordinate the consistency level dynamically and ensure the security of its data.**

Key Words : Tunable consistency; consistency level; replication factor; Cassandra

I. Introduction

In the context of massive data, the use of distributed database system has been an inevitable choice. The different distributed database requests have different consistency requirements. It is an ideal strategy to distinguish consistency dynamically for different requests [1]. Such a distributed database system can meet the requirements of consistency and reduce the delay.

The consistency strategy of the distributed database needs to meet the CAP rule [1]. The distributed database must ensure its partition tolerance. In order to make the application achieve the desired level, a trade-off between the availability and consistency of the entire system needs to be set [2].

The consistency is divided into strong consistency and weak consistency. The strong consistency sacrifices the availability. The weak consistency, which is represented by the final consistency, sacrifices the time of implementing the consistency. The requirements of the traditional strong consistency bring the problem of increasing delay, and make database services develop towards the weak consistency gradually. For example , the Distribute Relational Database Service (DRDS) in Alibaba emphasizes the ultimate consistency, rather than strong consistency. The DRDS endured the "double 11" visits and also ensured high availability and consistency [3].

Cassandra database has achieved the division of consistency at the operational level [4]. Cassandra allows users to use different conformance.In the literature [5], the author designed a data-centric approach to programming to achieve a tunable consistency.In the literature [6], the author proposed the TACT pattern, developed a conti-based continuous coherence pattern. In the literature [7], the author presented the IDEA model.

The TCS introduces the harmony algorithm [8] to calculate the number of replication required for different requests, which is used to get the consistency level. The TCS adds a Request Scheduling Layer(RSL) between the interface layer and the business logic layer to schedule requests according to the consistency level. There are three consistency priority queues in the RSL, which are used to schedule requests.

The experimental results show that the TCS can coordinate the consistency level dynamically and ensure the security of its data.

This paper is organized as follows. Section I introduces the TCS. Section II presents the experiment. Section III gives the experimental results. Section IV summarizes the conclusion.

II. Tunable Consistency Strategy(TCS)

A. Overall design framework of the strategy

Commonly used systems divide business applications into three layers: User Interface Layer, Business Logic Layer and Data access Layer. The TCS adds the RSL between the interface layer and the business logic layer.

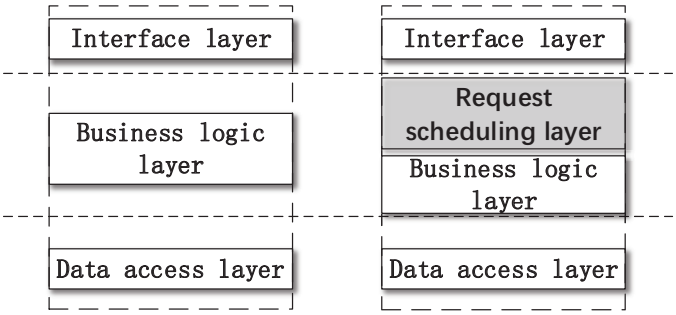

Fig. 1: The comparison of the structure before and after adding the tunable consistency strategy.

The RSL added in the TCS is the core architecture. The RSL is the consistent control layer and the implementation of the TCS. The TCS receives the data from the client and performs the consistency level processing, processes the data from the business logic layer and outputs it according to the consistency level.

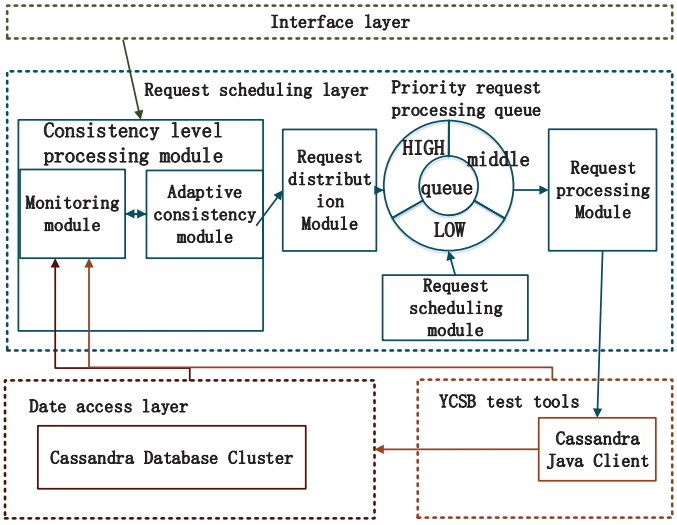

Fig. 2: System Module diagram

As shown in Figure 2, the RSL contains the consistency level processing module, the request distribution module, the priority request processing queue, the request scheduling module and the request processing module.

The consistency level processing module is divided into the monitoring module and the adaptive consistency module. The monitoring module is responsible for collecting the relevant information from the Cassandra storage cluster by the Cassandra Node tool and passing the information to the adaptive consistency module. The adaptive consistency module uses the harmony algorithm, and calculates how many replications of the request are processed to achieve the requirement of the application for the system consistency level.

The request distribution module calculates the threshold for real-time high, medium, and low queues based on the consistency level of each request obtained by concurrency. And the received requests are placed into the three queues according to the threshold.

The priority request processing queue consists of three queues. These queues determine which requests are stored in the queue, based on the working mechanism of the requesting distribution module. The requests in the queue are processed in the order of the high queue, the middle queue, and the low queue.

The request processing module reads the request from the three queues according to the priority and passes the request's corresponding consistency level to the Cassandra Java Client in the YCSB [9] module, while the module is responsible for passing the read data to the interface layer.

The Cassandra Java Client is responsible for reading the number of replications that are consistent with the request consistency level from the data access layer.

B. Realization of the strategy

The general idea of the TCS can be shown as Alg.1

Algorithm 1: TCS

```
1  QueHigh = newList();
2  QueMiddle = newList();
3  QueLow = newList();
4  X_n = Calconsistence(T_p, app_stale_rate, λ_r, λ_w);
5  // get consistency level based on Alg.2;
6  T_h = Calthreshold()// calculate threshold;
7  PushQue(T_h, X_n);
8  if (time > ΔT_1); then
9      Pop(QueMiddle, N_1)); // prevent starvation
10 else
11     if (time > ΔT_2); then
12         Pop(QueLow, N_2);// prevent starvation
13     else
14         DealQue(); // process queue normally
```

X_n is the number of replication; T_p is the time required to write or update the request to all copies; app_stale_rate (ASR)is the need of the application for consistency; λ_r and λ_w are the parameters of the exponential distribution followed by the random variable read time and write time; T_h is the threshold; N_1 and N_2 are the number of replications; ΔT_1 and ΔT_2 are the interval of the time for initialization.

1) system flow: This strategy designs three priority task queues of HQ (High priority task queue), MQ (Middle priority task queue) and LQ (Low priority task queue). This strategy creates a counter CO in each queue to hold the total number of requests in the current queue and needs to create an accumulator AC (Accumulator) to record the sum of the consistency levels of all requests currently in the queue.

a The module creates empty queues HQ, MQ, LQ, and initializes counter CO and accumulator AC to zero.

b When the consistency level processing module receives the request, the monitoring module will collect information related to the data from Cassandra Node tool and pass the information to the adaptive consistency module.

c After receiving the information, the adaptive consistency module will calculate the consistency of the request level by the harmony algorithm.

d After receiving the information, the adaptive consistency module will calculate the consistency of the request level according to the harmony algorithm. The request distribution module distributes requests to their corresponding queues and passes the consistency level to the corresponding HQ, MQ, LQ queues. Then the system updates the counters and accumulators, so that the threshold between HQ, MQ, and LQ are dynamically updated.

e After receiving the information, the adaptive consistency module will calculate the consistency of the request level by the harmony algorithm. The request processing

module reads X_n replications from Cassandra database according to the order.

f Repeat steps b-e until the database excepts or stops the service.

2) Consistency level processing module: In the adaptive consistency module, we introduce the harmony algorithm to get the consistency level of the request [10]. This algorithm uses stale read rate [11] to precisely define the consistency requirements of the application. The ASR describes the application's need for consistency. In order to meet the consistency requirements of the application, the condition ASR $\geq \theta_{stale}$ should be satisfied ,where θ_{stale} is the stale read rate. The main algorithm of Consistency level processing is as follows:

Algorithm 2: Consistency level processing

```
if (ASR ≥ θ_stale); then
    Choose eventuall consistency (Consistency Level = One);
else
    Compute X_n the number of always consistent replicas necessary to have ASR ≥ θ_stale;
Choose consistency level based on X_n;
```

a Calculate stale read rate
The process of arriving is often seen as a poisson distribution, which is used in both literatures [12] and [13] to describe transactional access.
The formula of stale read rate is as follows:

$$P_{stale\_read} = \frac{(N-1)(1-e^{-\lambda_r T_p})(1+\lambda_r\lambda_w)}{N\lambda_r\lambda_w} \quad (1)$$

b Calculate the number of replications
In order to meet the consistency requirements of the application(ie $ASR \geq \theta_{stale}$), we need to calculate the number of replications for the read requests(ie X_n).
X_n satisfy the following inequality:

$$X_n \geq \frac{N((1-e^{-\lambda_r T_p})(1+\lambda_r\lambda_w) - ASR\lambda_r\lambda_w)}{(1-e^{-\lambda_r T_p})} \quad (2)$$

3) Request distribution module: The formula for the threshold of the priority queue:

$$Th_{low} = \frac{\sum_{i=1}^{n} err\_val_i}{2n} \quad (3)$$

$$Th_{high} = \frac{2\sum_{i=1}^{n} err\_val_i}{n} \quad (4)$$

Th_{low} is the threshold of the current low-priority queue; Th_{high} is the threshold of the current high-priority queue; err_val_i is the consistency level of the i-th task in the priority request processing queue(ie, the value in the current accumulator); N is the total number of requests in current queue (ie, the current counter value).

The request distribution module obtains each request and its corresponding consistency level from the consistency level processing module. The module compares the consistency level with the current Th_{low} and Th_{high}, and the request is delivered to the corresponding consistency according to the result of the comparison queue. The processing flow of the priority request queue is as follows:

a If the consistency level of new request is less than Th_{low}, the request dispatch module will add the request to the end of the low priority queue.

b If the consistency level of the new request is less than or equal to Th_{high}, and greater than or equal to Th_{low}, the request distribution module will add the request to the end of the priority queue.

c If the new request's consistency level is greater than Th_{high}, the request distribution module will add the request to the end of the high priority queue.

d Counter CO is incremented by 1. The accumulator AC adds a new request for the consistency level, and the threshold is updated.

e Repeat step a-e, until the database excepts or stops the service.

4) Request processing module: The flow of request processing is as follows:

a The request processing module estimates whether the high priority queue is empty. If it is not empty, then takes out a request to perform from the header of the high priority queue.

b If the high priority queue is empty, then it determines if the priority queue is empty. If it is not empty, the module gets a request from the header of the middle priority queue.

c If the priority queue is empty, then it is judged whether the low priority queue is empty. If it is not empty, then the module gets a request from the head of the low priority queue to execute.

d If the low priority queue is empty, then go to step 2-4, until the database stops the service so far.

e After removing the request to be executed, the counter CO is decremented by one. The accumulator AC subtracts the consistency level of the request, and the threshold is updated.

f Repeat steps a-e, until the database excepts or stops the service.

III. Experiment and analysis

A. Introduction to the experimental environment

In the cassandra experimental platform, we set server 2 as a seed node, and server 9 as a client access. The role of seed nodes is that when a new node is added to a cluster, it is necessary to run the gossip process. This experiment is tested using the YCSB tool, and the YCSB is called "Yahoo! Cloud Serving Benchmark".

The experimental environment is as follows : the Python version is $2.7.12+$; the JDK version is $1.8.0\_111$; the Apache Maven version is 3.3.9; the Cassandra database cluster version is the latest version 3.9; the test tool YCSB version is 0.13.0.

The experiment in this paper uses YCSB's default five loads for experimentation, as shown in Table 1:

TABLE I: Workloads used in the experiment

load	settings	Choose operation	example
A-Read and Update	read50% update50%	Zipfian	Shopping cart online
B-Read mostly	read95% update5%	Zipfian	Photo tag
C-Read latest	read80% insert20%	Latest	User status read and update
D-Read-modify-write	read50% update50%	Zipfian	User profile cache
E-Scan short ranges	select95% insert5%	Zipfian / Uniform	Thread session

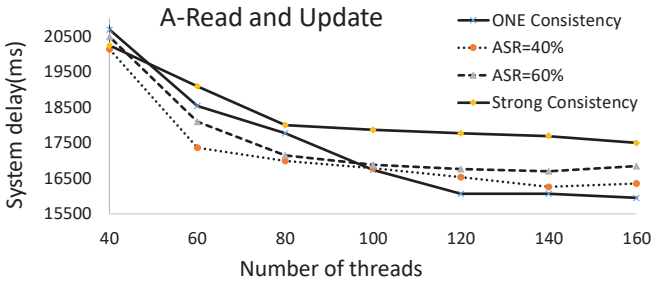

Fig. 3: The relationship between client threads and system delay(a)

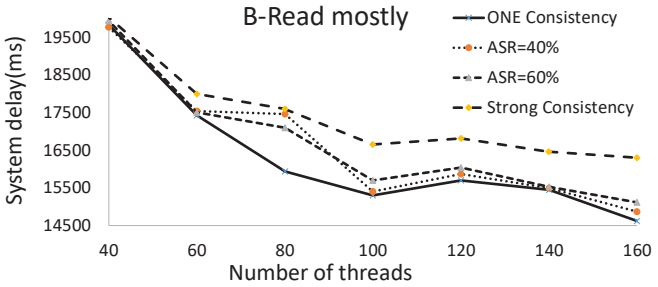

Fig. 4: The relationship between client threads and system delay(b)

B. *Experimental process and parameter setting*

The experiments tested the five workloads in Table 1 and loaded the data into eight Cassandra copies with a total data size of 4.8 GB. The experiments were conducted with two different ASR values and were compared with the ONE consistency strategy and strong consistency. The first ASR value is chosen to be 60%. The value shows that its consistency requirement is slightly lower and the waiting time is shorter. The second ASR value is chosen to be 40%, which is slightly more stringent than the first ASR value, indicating that the request may read more copies and the consistency requirements are higher.

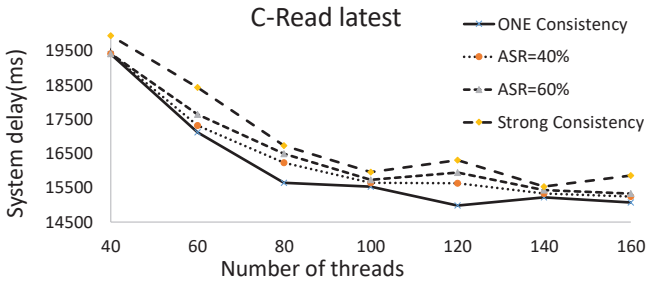

Fig. 5: The relationship between client threads and system delay(c)

C. *results and analysis*

1) Delay experiment: The number of threads in experiments are 40, 60, 80, 100. We tested the delay time for different consistency cases under the five workloads.

It can be seen from Figure 3, 4, 5, 6, 7, that the strong consistency has the longest latency in any case, since all replicas are required to return the result. The ONE consistency strategy provides the shortest latency scenario because it requires only one copy of the return value, but the results are in a lack of consistency in the large amount of data.ASR is inversely proportional to the delay time. Overall, the time delay of the TCS is basically between the ONE consistency strategy and the strong consistency delay.

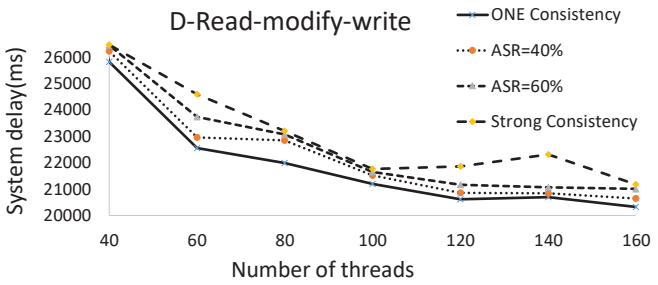

Fig. 6: The relationship between client threads and system delay(d)

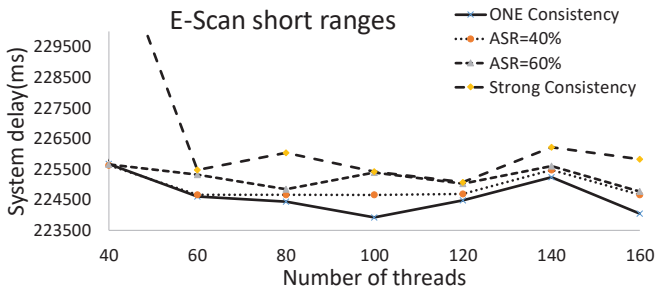

Fig. 7: The relationship between client threads and system delay(e)

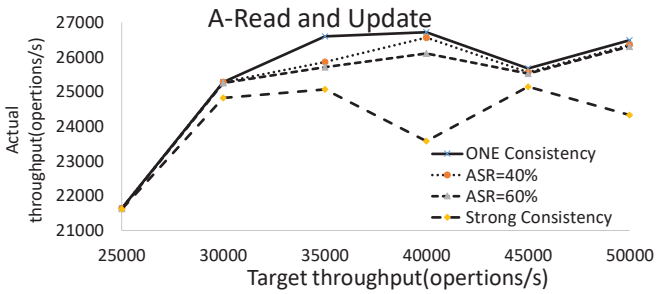

Fig. 8: The relationship between target throughput and actual throughput(a)

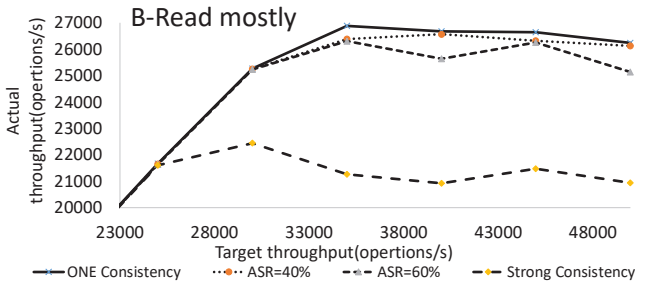

Fig. 9: The relationship between target throughput and actual throughput(b)

2) Throughput experiment: It can be seen from Figure 8, 9, 10, 11, 12, that the actual throughput increases as the target throughput increases. When the target throughput reaches a certain amount, the increase in actual throughput tends to be stable or even reduced. It is because the throughput has reached the limits of the system. The throughput of the TCS is greater than the strong consistency and less than the ONE coherency strategy, since the higher the consistency requirement is, the longer the time it will take to process a single request, so the number of requests in unit time can be reduced. Regardless of whether the ASR is set to 40% or 60%, the throughput is less than the ONE consistency strategy in workloads A, B, D, and E. But in most cases ASR = 60% throughput is still less than ASR = 40% throughput.

3) Read data exception experiment: Since the YCSB does not count stale read rate, the experiment first outputs the

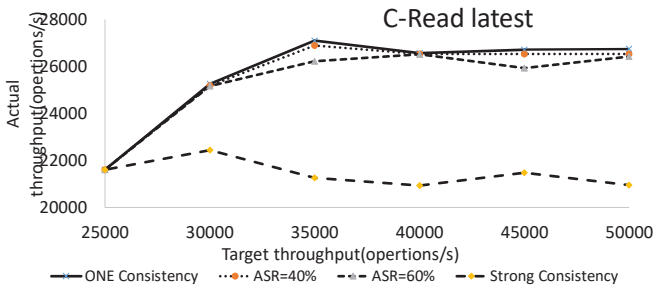

Fig. 10: The relationship between target throughput and actual throughput(c)

Fig. 11: The relationship between target throughput and actual throughput(d)

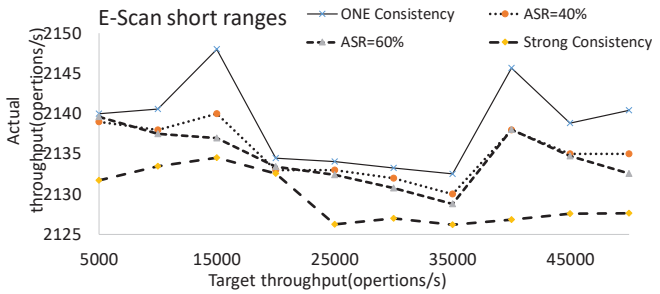

Fig. 12: The relationship between target throughput and actual throughput(e)

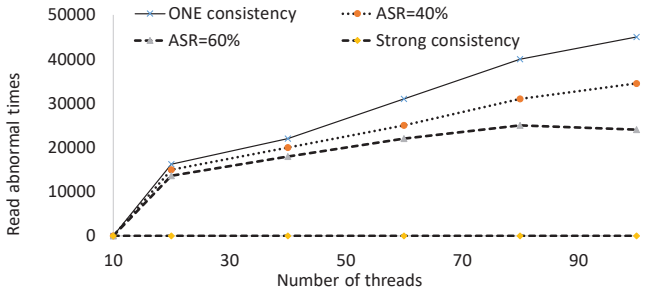

Fig. 13: The relationship between client threads and number of stale reads

results of the strong consistency and assumes that all data are correct(ie $\theta_{stale} = 0$). The experiment compared the ASR = 40%, ASR = 60%, and ONE consistency strategies with strong

consistency as benchmarks. The experiment uses the workload A-Read and Update.

It can be seen from Figure 4, that the stale read rate is less than that of the ONE consistency strategy. And the stale read rate becomes higher when the ASR becomes less. When the number of threads is higher than 40, the stale read rate in the TCS has a tendency of decreasing. And the gap between the ONE consistency strategy and the TCS also increases.

IV. Conclusion

In the circumstances of the big data era, the scalability and availability of applications are more stringent. The availability of strong consistency was unable to meet the requirements of applications. Therefore, the weak consistency strategies are proposed. But on the other hand, the weak consistency will lead to a lot of inconsistent data in applications. TCS was proposed in this paper.

First, the tools collect time latency, the number of ASR and the information of write and read requests. And in the Harmony algorithm the number of copies of the database needing to be read is calculated, in order to achieve the probability which the user set. Higher priority is given to heavy operations, therefore a operation having a higher level of consistency can be handled first. And a number of modules are built.

According to the experiments, compared with the traditional strong consistency strategy, the TCS introduced in this paper has lower time latency and higher throughput in the most of workloads in different threads. In addition, it is possible to provide a more appropriate level of consistency than the ONE consistency. And the probability of the stale read happened is lower than ONE strategy.

References

[1] G. Harrison, *Next Generation Databases*. Apress, 2015.

[2] F. Wang and G. Zhong, "Research on technology of drds distributed database," in *International Conference on Information Technology and Management Innovation*, 2015.

[3] Techweb, "Decrypt the ali cloud enterprise-class internet architecture behind double eleven," 2015.

[4] H. E. Chihoub, S. Ibrahim, Y. Li, G. Antoniu, M. S. Perez, and L. Bouge, "Exploring energy-consistency trade-offs in cassandra cloud storage system," in *International Symposium on Computer Architecture and High PERFORMANCE Computing*, 2015, pp. 146–153.

[5] N. Zaza and N. Nystrom, "Data-centric consistency policies: A programming model for distributed applications with tunable consistency," in *The Workshop on Programming MODELS and Languages for Distributed Computing*, 2016, p. 3.

[6] H. Yu and A. Vahdat, "Design and evaluation of a conit-based continuous consistency model for replicated services," *Acm Transactions on Computer Systems*, vol. 20, no. 3, pp. 239–282, 2002.

[7] Y. Lu, Y. Lu, and H. Jiang, "Adaptive consistency guarantees for large-scale replicated services," in *International Conference on Networking, Architecture, and Storage*, 2008, pp. 89–96.

[8] H. E. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Perez, "Harmony: Towards automated self-adaptive consistency in cloud storage," in *IEEE International Conference on CLUSTER Computing*, 2012, pp. 293–301.

[9] A. Dey, A. Fekete, R. Nambiar, and U. Rohm, "Ycsb+t: Benchmarking web-scale transactional databases," in *IEEE International Conference on Data Engineering Workshops*, 2014, pp. 223–230.

[10] H. Liang, J. Wei, Y. Xing, and J. Wu, "A feature-based replica consistency strategy for spatial data in distributed gis," in *International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2010, pp. 11–16.

[11] H. E. Chihoub, S. Ibrahim, G. Antoniu, and M. Prez, "Consistency management in cloud storage systems," *Large Scale and Big Data - Processing and Management*, 2014.

[12] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu, "Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective," in *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, 2011, pp. 134–143.

[13] A. T. Tai and J. F. Meyer, "Performability management in distributed database systems: An adaptive concurrency control protocol," in *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, 1996, p. 212.